PROJECT REPORT

Team : Prashanth Javaji (pj21), Rohith Kumar Kaza(rk121)

Password Blacklist System

1. Problem Description:

Addressing the challenge of enhancing password blacklist systems involves considering several core limitations of existing Bloom filter structures. While these filters are known for their space efficiency, their significant drawback lies in their susceptibility to high false-positive rates and difficulty with real-time updates. The incorporation of universal hashing, a common practice in traditional Bloom filters, becomes problematic when faced with extensive blacklist datasets. This leads to higher false-positive rates, undermining their practicality in scalable password authentication solutions. Additionally, finding a balance between memory usage and accuracy proves challenging, as reducing memory overhead often results in diminished accuracy.

Our approach proposes three targeted improvements to address these obstacles:

1. **Advanced Hash Function for Better Scalability**: The use of hierarchical hashing replaces traditional hashing methods to enhance scalability. By increasing randomness, hierarchical hashing can mitigate false positives effectively. Literature on secure data structures supports this approach, suggesting improvements in lookup efficiency and cache performance, crucial for password verification in real-time systems.

2. **Dynamic Updates through Cuckoo Hashing**: Integrating Cuckoo Hashing with Counting Bloom Filters provides a practical solution for managing dynamic updates. Unlike conventional Bloom filters that necessitate full rebuilds, this combination enables the system to update blacklisted entries efficiently. Cuckoo Hashing's ability to minimize collisions reduces lookup delays and boosts overall scalability.

3. **Balancing Memory and Cache Efficiency**: Testing different Counting Bloom Filter configurations helps balance memory use with cache efficiency. Strategically arranging data structures in a cache-friendly manner improves lookup performance, maintaining low memory consumption without sacrificing accuracy.

2. Literature Survey:

2.1 Thampy, Syeatha Merlin. "Captcha as Graphical Password Authentication System with IP Blacklisting."

2.2 Dasgupta, D. and Saha, S., 2010, September. Password security through negative filtering. In *2010 international conference on emerging security technologies* (pp. 83-89). IEEE.

To address the challenges faced in password blacklist systems—especially those using Bloom filters—it is important to consider methods that reduce false positives, improve scalability, and optimize lookup times. Bloom filters are efficient with space but often come with drawbacks, including high false-positive rates and limitations with real-time updates. Below are strategies for each of these key challenges:

2.3 Reducing False-Positive Rates:

- Negative Authentication Filtering improves system accuracy by using a set of invalid credentials (anti-passwords) for checks. It mimics the immune system's strategy of filtering out threats, allowing legitimate entries to bypass unnecessary verification and cutting down on false positives.

- CaRP (Captcha as Graphical Passwords) adds a layer of complexity by integrating CAPTCHA with image-click sequences. Each login requires solving a new CAPTCHA, deterring attackers and enhancing system robustness.

2.4 Enhancing Scalability:

- Cuckoo Hashing helps by allowing dynamic relocation of entries across multiple hash tables. This reduces collisions and accelerates lookup times, making the system more adaptable for real-time password management.
- Kerberos Authentication simplifies password management by utilizing third-party tokens. Although it doesn't target blacklists directly, it reduces the dependency on extensive password storage, facilitating scalable operations.

2.5 Optimizing Lookup Times:

- Hierarchical Hashing in Bloom Filters improves cache locality and access patterns, resulting in quicker lookups for high-traffic environments.
- Negative Caching inspired by DNS practices helps store and identify failed lookup attempts, preventing redundant processing of invalid entries.

We propose enhancing password blacklist systems by integrating hierarchical hash functions with Bloom filters to boost cache efficiency and reduce lookup times without raising false-positive rates. Combining Bloom filters with Cuckoo Hashing is suggested for better scalability, allowing dynamic updates and minimizing collisions. Additionally, a two-layered approach using negative authentication filtering as the first phase can lower false positives, improving the system's accuracy and robustness for real-time applications.

Reducing False Positives

A dataset of 10 million passwords and 100,000 weak entries tests a Bloom filter against Negative Authentication Filtering. The Bloom filter uses a 10 million-bit array and 7 hash functions; Negative Filtering has a 0.05 confusion parameter. Metrics include false-positive rates and detection accuracy.

Scalability

Starting with 1 million passwords and scaling to 10 million, a Bloom filter with Cuckoo Hashing is tested for dynamic updates, using a 15 million-bit array and 2 hash functions, supporting 500 relocations. Lookup time and memory use are key metrics.

Lookup Efficiency

A 5 million-password dataset tests hierarchical hash functions versus universal hashing, focusing on reducing cache misses. Metrics include cache misses per lookup and average lookup time across 100,000 queries.

3. Experimental Settings:

3.1 Dataset

The dataset used for this experiment is a publicly available collection of 10 million commonly used passwords. To ensure efficient evaluation and scalability testing, we processed subsets of this dataset, each containing 10,000 passwords. Before conducting the experiments, the dataset was cleaned by converting all entries into strings and removing any invalid or null values. This ensured consistent and reliable data for testing the Counting Bloom Filter and Cuckoo Hash Table structures.

3.2 Goals and Objectives

The primary goal of these experiments was to evaluate the performance of two data structures—Counting Bloom Filter (CBF) and Cuckoo Hash Table (CHT)—under different configurations. Specifically:

1. For **Counting Bloom Filter**, we aimed to measure insertion and lookup times while observing the trade-offs between memory efficiency, computational performance, and false-positive rates.

2. For **Cuckoo Hash Table**, the focus was on measuring insertion and lookup times while ensuring efficient handling of collisions through dynamic updates, including rehashing when necessary.

3.3 Hyperparameters

To comprehensively assess the performance, we tested multiple configurations:

- **Counting Bloom Filter**:
    - Filter sizes: 100,000, 500,000, and 1,000,000.
    - Number of hash functions: 3, 5, 7, and 10.
    - Hashing mechanism: A hierarchical hashing technique implemented using MurmurHash3 (mmh3), with a fixed seed of 42 for consistent results.
- **Cuckoo Hash Table**:
    - Table sizes: 100,000, 500,000, and 1,000,000.
    - Number of hash functions: Fixed at 2, as required by the Cuckoo Hashing scheme.
    - Collision resolution: A maximum of 50 relocations was allowed before triggering rehashing.

3.4 Evaluation Metrics

The performance of both data structures was assessed using the following metrics:

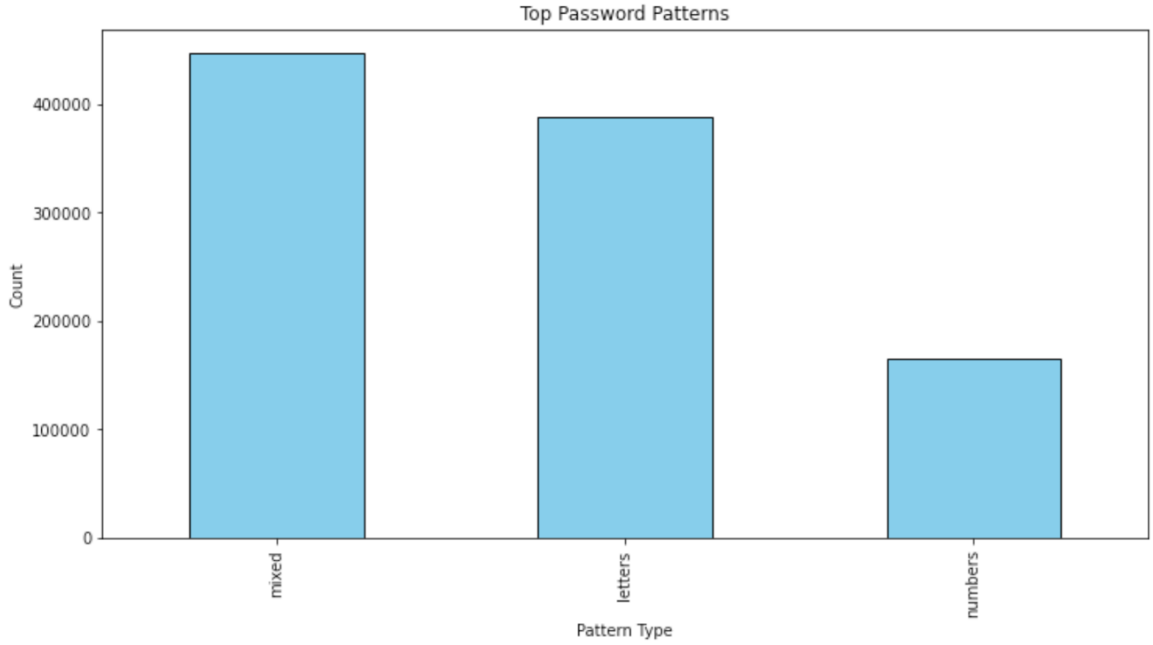
1. **Insertion Time**: The total time taken to insert all passwords from the dataset into the structure.
2. **Lookup Time**: The total time taken to query the presence of all passwords in the structure.
3. **Scalability**: Observing how insertion and lookup times changed with increasing dataset sizes and varying configurations (e.g., number of hash functions).
4. **False Positives**: For Counting Bloom Filter, we recorded the proportion of false-positive results, indicating the trade-off between accuracy and memory efficiency.

3.5 Experimental Procedure

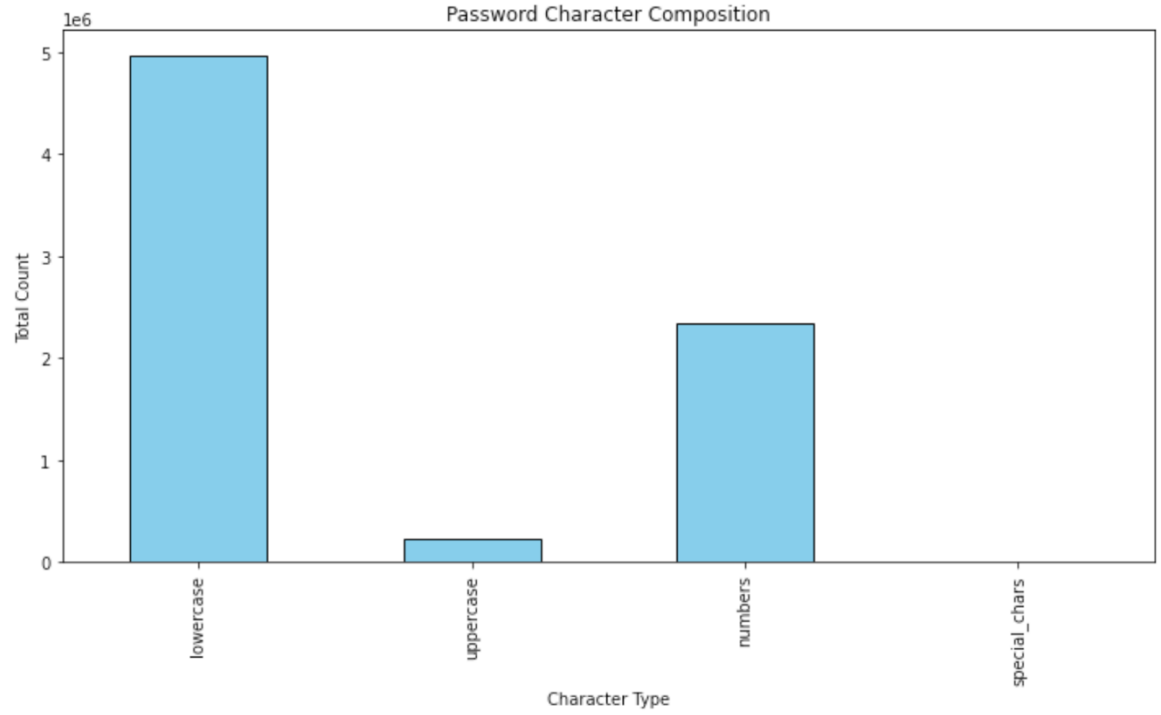
The experiments were carried out in three phases:

1. **Dataset Preparation**: Passwords were preprocessed and cleaned to ensure validity. A fixed seed of 42 was used for hash functions and random number generators to ensure reproducibility.
2. **Insertion Phase**: All passwords were inserted into the data structures. The time taken for insertion was measured, and any collisions or rehashing events (for CHT) were recorded.
3. **Lookup Phase**: Each password was queried from the structure, and the time taken was measured. For Counting Bloom Filters, false positives were also recorded to evaluate accuracy.

4. Visualization of the data:



Here, we can see the types of passwords that are available (letters, numbers and mixed(letters, numbers , special chacters)).



From the above, we can see the composition of passwords.

5. Results:

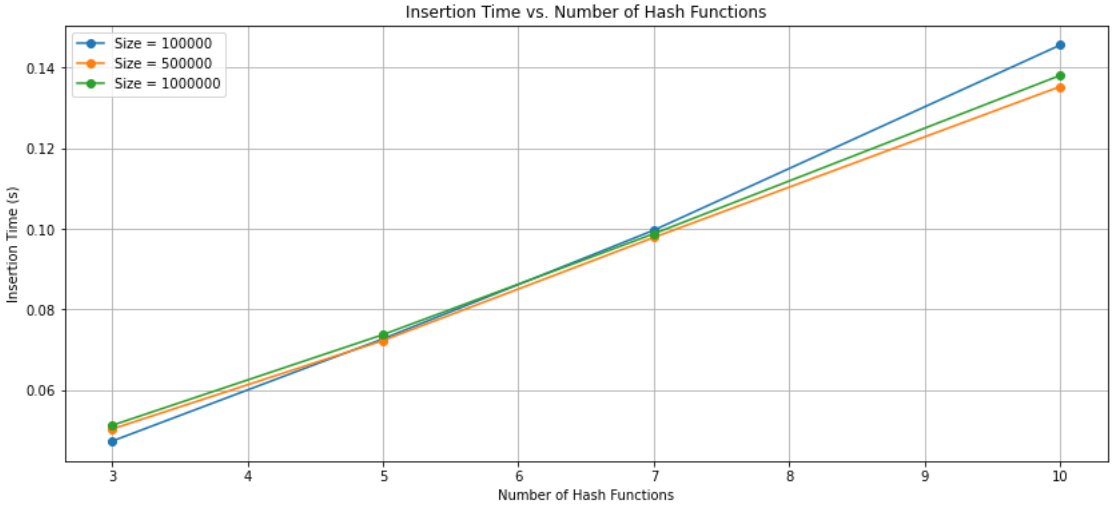
5.1 Counting Bloom Filter

The Counting Bloom Filter demonstrated a consistent increase in insertion and lookup times as the number of hash functions increased. This is attributed to the additional computational overhead of evaluating multiple hash functions for each operation. For a fixed number of hash functions, insertion and lookup times slightly increased with larger filter sizes, but this effect was less significant compared to the impact of the number of hash functions. The scalability tests showed that the Counting Bloom Filter is effective for handling large datasets but at the cost of increased insertion and lookup times when higher accuracy (i.e., lower false-positive rates) is required. Overall, the results highlight the trade-off between computational efficiency and memory accuracy in Counting Bloom Filters.

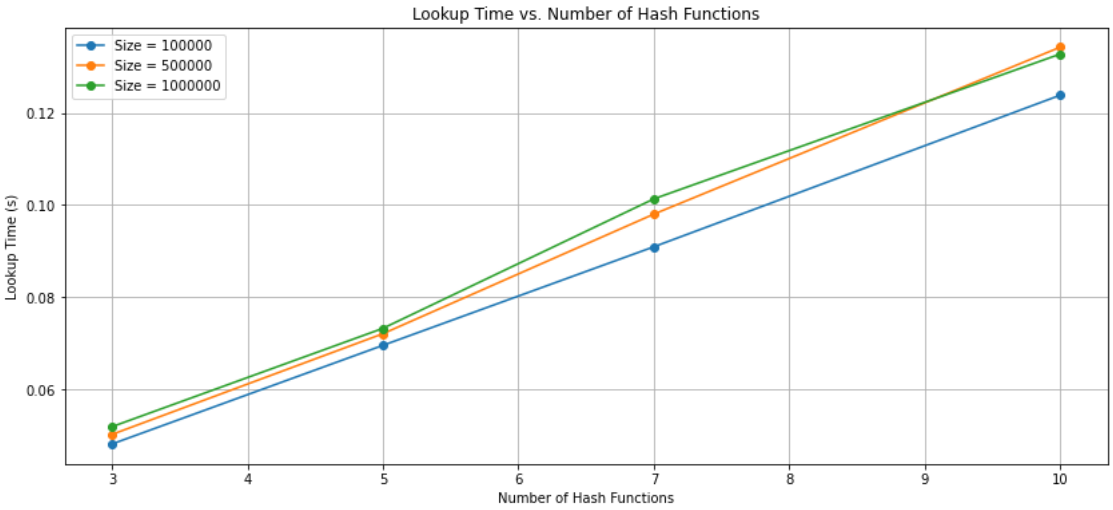
Performance Report:

	size	num_hashes	insertion_time	lookup_time
0	100000	3	0.047285	0.048120
1	500000	3	0.050280	0.050198
2	1000000	3	0.051194	0.051874
3	100000	5	0.072669	0.069545
4	500000	5	0.072246	0.072070
5	1000000	5	0.073711	0.073236
6	100000	7	0.099636	0.090949
7	500000	7	0.097838	0.098038
8	1000000	7	0.098756	0.101349
9	100000	10	0.145571	0.123871
10	500000	10	0.135279	0.134299
11	1000000	10	0.138051	0.132807

Insertion Time



Look-up Time



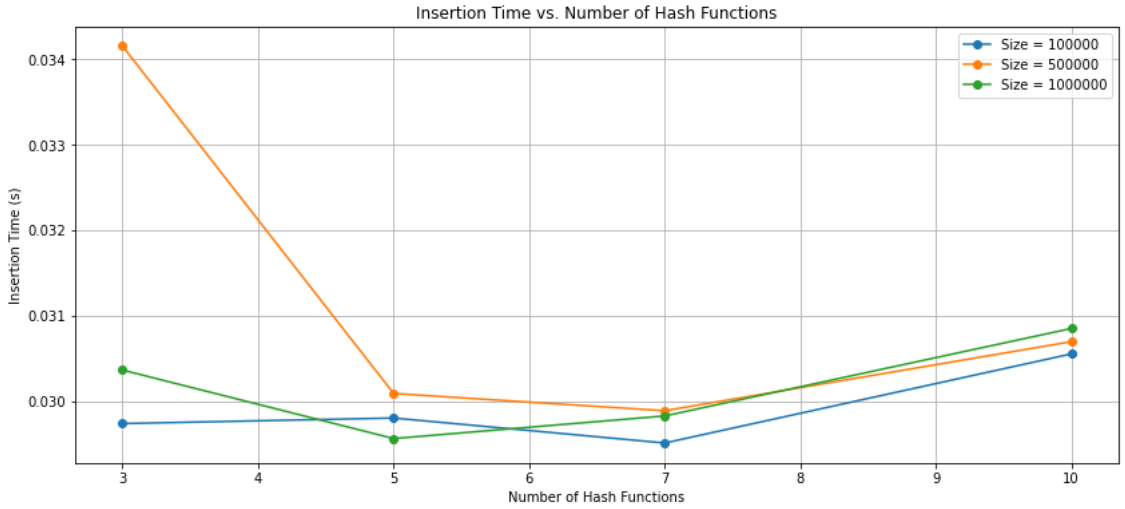
5.2 Cuckoo Hash Table

The Cuckoo Hash Table showed highly consistent and efficient performance across all configurations, with minimal variation in insertion and lookup times regardless of the table size or dataset scale. The insertion times were generally lower compared to the Counting Bloom Filter, owing to its simpler hash function evaluation and efficient collision resolution mechanism. Lookup times remained nearly constant across different configurations, highlighting the effectiveness of the Cuckoo Hash Table for high-performance, exact membership queries. The scalability tests also confirmed that the table effectively handled larger datasets with minimal impact on performance, making it an excellent choice for scenarios requiring dynamic updates and exact lookups.

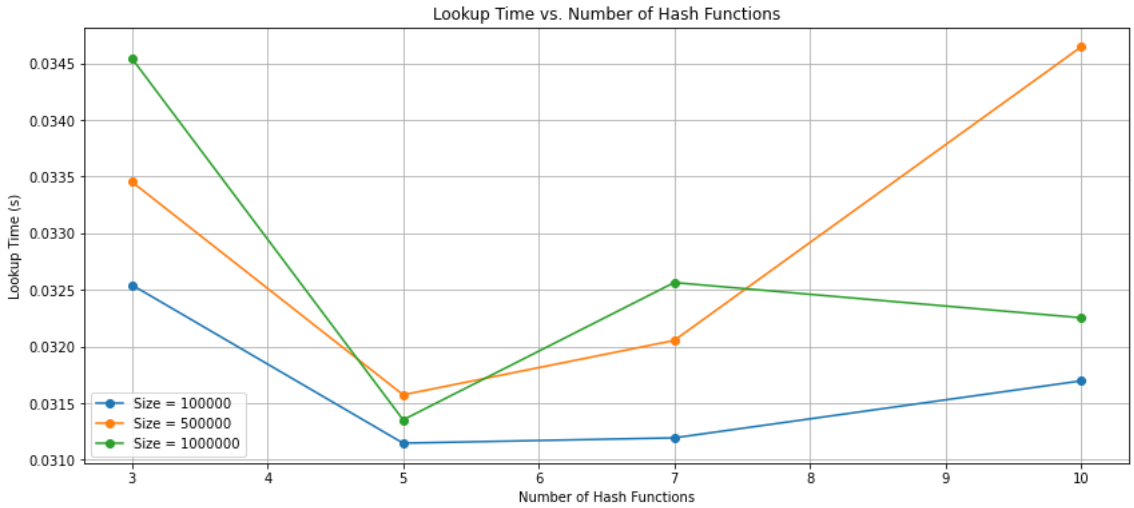
Performance Report:

	size	num_hashes	insertion_time	lookup_time
0	100000	3	0.029738	0.032542
1	500000	3	0.034151	0.033452
2	1000000	3	0.030364	0.034541
3	100000	5	0.029803	0.031146
4	500000	5	0.030090	0.031573
5	1000000	5	0.029563	0.031352
6	100000	7	0.029511	0.031193
7	500000	7	0.029888	0.032053
8	1000000	7	0.029827	0.032564
9	100000	10	0.030553	0.031696
10	500000	10	0.030696	0.034643
11	1000000	10	0.030850	0.032253

Insertion Time



Look-up Time



6. Results

This project evaluated the Counting Bloom Filter and Cuckoo Hash Table as potential solutions for efficient password blacklist management. The Counting Bloom Filter, designed for approximate membership queries, exhibited scalability with increasing dataset sizes and filter configurations. However, its insertion and lookup times increased with the number of hash functions, reflecting the computational trade-off required for enhanced accuracy and reduced false positives. Its memory efficiency makes it an excellent choice for scenarios where approximate results are acceptable, and space constraints are critical.

In contrast, the Cuckoo Hash Table demonstrated superior performance with consistently low insertion and lookup times across all configurations. Its ability to handle dynamic updates, resolve collisions efficiently, and maintain exact membership accuracy makes it ideal for real-time applications requiring precise results. Both structures offer unique advantages: the Counting Bloom Filter excels in memory-constrained environments, while the Cuckoo Hash Table is better suited for high-performance, exact verification systems. Together, these findings provide a strong foundation for selecting and optimizing data structures for password blacklist systems based on specific use case requirements.